Summary: *CURE: An Efficient Clustering Algorithm for Large Databases.* (2001) Sudipto Guha (Stanford), Rajeev Rastogi (Bell Labs), Kyuseok Shim (Korea Institute of Technology)

**Main point:** A new scalable algorithm called CURE is introduced, which uses random sampling and partitioning to reliably find clusters of arbitrary shape and size. CURE clusters a random sample of the database in an agglomerative fashion, dynamically updating a constant number $c$ of well-scattered points $R_1, \ldots, R_c$ per cluster to represent each cluster’s shape. To assign the remaining, unsampled points to a cluster, these points $R_i$ are used in a similar manner to centroids in the $k$-means algorithm – each data point that was not in the sample is assigned to the cluster which contains the point $R_i$ closest to the data point. To handle large sample sizes, CURE divides the random sample into partitions which are pre-clustered independently, then the partially-clustered sample is clustered further by the agglomerative algorithm. A heap is used to keep track of which clusters to merge at each iteration, and a $kd$-tree is used to find the closest representative point to a given point $x$. The time complexity of CURE is $O(n^2 \log n)$ in general, and in the case when the data points lie in a two-dimensional space, the time complexity can be shown to reduce to $O(n^2)$.

The authors begin by pointing out several disadvantages of existing clustering algorithms, such as the inability of $k$-means to find clusters that are not in the shape of a hyper-sphere and the biases of the large-scale clustering algorithms BIRCH and CLARANS for uniformly-sized clusters which are convex or spherical. The sensitivity of the DBSCAN algorithm to outliers and to the user-specified density parameters $\epsilon$ and $MinPts$ is pointed out, as is its inability to scale and the possible chaining effect of the algorithm.

A new algorithm for detecting arbitrarily-shaped clusters at large-scale is presented and named CURE, for “Clustering Using Representatives”. The algorithm works by pre-clustering a sample of the entire dataset, then using representative points within the sample to assign the remainder of the dataset. The sample is clustered using an agglomerative clustering algorithm which keeps track of the representative points in each cluster, as well as the nearest neighbor of each cluster at each step.

The goal of CURE is to detect clusters of arbitrary shapes and sizes, such as those illustrated in Figure 1. In order to detect these clusters, the algorithm proceeds as follows.

First, a random sample is drawn from the database of points, as shown by the orange nodes in Figure 2.

This smaller subset of random points, labeled by node number in Figure 3 is then passed into the hierarchical clustering algorithm.

The clustering algorithm first considers each point to be its own cluster, and identifies the nearest neighbor of each cluster, as drawn by the blue lines connecting nearest neighbors in Figure 4.

A heap is used to keep track of the nearest two clusters in any iteration.
Figure 1: Example of clusters with arbitrary shapes and sizes in 2-D, detectable by CURE.

Figure 2: Example of a random sample of points drawn from the entire dataset in the first step of CURE.
Figure 3: Labeled data points in the sample drawn by CURE in the first step of the algorithm.

Figure 4: The nearest neighbor of each cluster is found – in the first step, each point is its own cluster.
The clusters are stored in the heap according to their distance to their nearest neighbor. In other words, cluster $u$ is stored according to its distance $d(u, u_{\text{nearest neighbor}})$ as shown for the first step of the algorithm in Figure 5. In the first step, $d(u, u_{\text{nearest neighbor}})$ is simply the distance between a node and its nearest neighbor.

When a cluster $u$ contains more than one node, then $d(u, u_{\text{nearest neighbor}})$ is determined by the distance between the nearest representative point in $u$ to a representative point in a different cluster $v$. The number of representative points $c$ is a user-defined input, and the points are chosen according to the following procedure: at each merge between clusters $u$ and $v$, the weighted mean between the means of these two clusters is chosen as the first representative point of the newly merged cluster $w = \text{merge}(u, v)$. The weighting is determined by the number of points in each cluster, so that a cluster with more points assigns more weight to its mean. Next, for each $i = 2, \ldots, c$, a node from the merged set $u \cup v$ becomes a representative point $R_i$ if it is the farthest point from the previous point $R_{i-1}$ in $u \cup v$. After all $c$ representative points are chosen from the merge of $u$ and $v$, they are each “shrunk” by a factor $\alpha$ toward the mean of the merged set. In other words, each representative point $R_i$ is rescaled by:

$$R_i = R_i + \alpha \ast (\text{mean}(u, v) - R_i).$$

The set of these points is then stored in $w_{\text{rep}}$.

Next, the nearest neighbors are re-computed for all clusters, using a $kd$-tree data structure. The $kd$-tree stores all the representative points and is used to find the closest point to a given node. An example of a 2$d$-tree is given in Figure 6.

The new nearest neighbor of a cluster $w$ can be found by finding the closest point $p$ in the tree which is not in $w_{\text{rep}}$. In otherwords, the nearest neighboring

Figure 5: A heap where the distance $d(u, u_{\text{nearest neighbor}})$ is used to sort the clusters $u$. 
cluster to cluster $w$ is the cluster satisfying:

$$\min_v d(R_i \in w.\text{rep}, R_j \in v.\text{rep})$$

This can be done in $\log m$ time for each point $R_i$ in $w.\text{rep}$ and $m$ points in the tree. The heap is then reordered according to the new nearest neighbors of each cluster.

To eliminate outliers from becoming representative points and destroying the accuracy of the algorithm, a pruning step is called after the number of clusters reaches a user-defined fraction of the initial number of clusters. In the experiments, this fraction was chosen to be $\frac{1}{3}$. At this point, clusters of very small size (1 or 2 points) are eliminated. In addition, a final stage of the algorithm also eliminates clusters of very small size when the number of clusters is down to some small constant times $k$.

The worst-case time complexity of the hierarchical algorithm is $O(m^2 \log m)$ for $m$ points in the random sample initially drawn by the algorithm. The random sample is chosen in one pass over the data of $n$ points, and “has the desirable effect of filtering outliers.” The sample size $m$ is chosen by using Chernoff bounds to compute the minimal number of points such that at least $f|u|$ points are selected from a cluster $u$, for $0 \leq f \leq 1$, with probability $1 - \delta$, $0 \leq \delta \leq 1$. The authors explain that given a number of clusters $k$, we expect the smallest size of a cluster to be $n\rho^k$, where $\rho > 1$ and $n$ is the number of point in the dataset. Taking this together with the assumption that we need a constant number of points $\xi$ to represent a cluster of a particular geometry, the authors state that we can find a sample size that is independent of the number of points in the data set, making the algorithm very scalable.

Figure 6: An example of a 2d-tree. The nodes are ordered according to their position in each of the $k$ dimensions, repeating dimensions after each $k$ levels. In this case, $k = 2$. 
In the case when the sample size \( m \) is too large to handle efficiently by the agglomerative clustering algorithm, a partitioning scheme is used to first pre-cluster partitions of the sample, then aggregate these clusterings into a clustering of the entire sample. In the pre-clustering stage, the sample of size \( m \) is divided into clusters of size \( \frac{m}{pq} \), where \( p \) is the number of partitions that can be divided among processors, and \( q > 1 \) is a constant. The points are pre-clustered by the following procedure: “each successive data point becomes part of the closest existing cluster if it is within some threshold distance \( \tau \) from it – else, it forms a new cluster.” Ensuring that \( \tau \) is small enough will ensure that \( \frac{m}{pq} \) is sufficiently larger than \( k \), the desired number of clusters and that no two clusters are merged which shouldn’t be. By running the hierarchical algorithm on these partitions independently, and storing only the representative points for each clustering, a second pass over the pre-clustered sample can efficiently cluster the entire sample by only operating on the representative points, which can will all into main memory.

In the final stage of the algorithm, each point \( x \) that wasn’t in the sample is assigned to the cluster which contains the representative point closest to \( x \).

CURE is compared to BIRCH and MST in the experimental results section. The number of representative points \( c \) used by CURE was set to 10, the sample size \( m \) to 2500, and the shrink factor \( \alpha \) to 0.3. In this comparison, CURE is found to be less sensitive to outliers than MST and successfully finds elongated clusters that BIRCH tends to split. CURE is also much faster than BIRCH even on datasets with spherical, compact clusters, which are ideal for the BIRCH algorithm, due to the effectiveness of the partitioning used in CURE. When examining the effect of the choice of parameter settings in CURE, the authors find that setting the shrink factor to a number in the range 0.2-0.7 worked very well and found the right clusters. Setting \( \alpha \) to small values \(< 0.2 \) made CURE behave like the MST algorithm, sensitive to outliers, and setting it to larger values \( > 0.7 \) caused CURE to act like a centroid-based algorithm, becoming unable to find clusters of arbitrary shape. The number of representative points affected cluster quality when this number was too low, but at 10 representative points or more, CURE successfully identified the correct clusters. For sample sizes of 2500 or more (2.5% of the original dataset), CURE found clusters with great success.